Machien Learning and Inductive Inference [H02C1a]

Xinhai Zou (r0727971)

Contents

1 Lecture 1: Introduction, Version spaces

1.1 Some ML examples in practice

1. Autonomous cars
2. The Robosail project
3. The Robot Scientist
4. Infra Watch, "Hoolandse brug" - the bridge
5. Language learning
6. Automating manual tasks

1.2 Machine Learning

Definition of machine learning: it is the study of how to make programs improve their performance on certain tasks from own (experience).
In this case:

- "performance" = speed, accuracy
- "experience" = earlier observations

Machine Learning vs. other AI
In **machine learning**, the key is **data**, examples of questions and their answer; observations of earlier attempts to solve the problem
In **inductive inference**, it is reasonsing from **specific** to **general**, statistics: sample -> population; from **concrete observations** -> **general theory**

1.3 Machine Learning learning landscape

- tasks
  - clustering
  - classification
  - regression
  - reinforcement learning
- techniques
  - Convex optimization
  - Matrix factorization
  - Transfer learning
  - Learning theory
  - Greedy search

- models
  - automata
  - neural network
  - deep learning
  - statistical relational learning
  - decision trees
  - support vector machines
  - nearest neighbors
  - rule learners
  - bayesian learning
  - probabilisitc graphical models
- applications
  - natural language processing
  - vision
  - speech
- related courses
  - neural computing
  - support vector machine
  - uncertainty in AI
  - data mining
  - genetic algorithms and evolutionary computing

1.4 Some basic concepts and terminology

- Predictive learning
  - Definition: learn a model that can predict a particular property/ attribute/ variable from inputs
  - Binary classification: distinguish instances of class C from other instances
  - Classification: assign a class C (from a given set of classes) to an instances
  - Regression: assign a numerical value to an instance
  - multi-label classification: assign a set of labels (from a given set) to an instance
  - multivariate regression: assign a vector of numbers to an instances

  - multi-target prediction: assign a vector of values (numerical, categorical) to an instances

- Descriptive learning
  - Definition: given a dataset, describe certain patterns in the dataset, or in the population it is drawn from

- Typical tasks in ML
  - function learning: learn a function X->Y taht fits the given data
  - distribution learning: distribution learning
    * parametric: the function family of the distribution is known, we only need to estimate its parameters
    * non-parametric: no specific function family assumed
    * generative: generate new instances by random sampleing from it
    * discriminative: conditional probability distribution

- Explainable AI (XAI)
  - Definition: means that the decisions of an AI system can be explained
  - Two different levels:
    * We understand the (learned) model
    * We understand the individual decision

1.5 Input formats (predictive learning)

- Set
  - training set: a set of examples, instance descriptions that include the target property (a.k.a. labeled instances)
  - prediction set: a set of instance descriptions that do not include the target property ('unlabeled' instances)
  - prediction task: predict the label of the unlabeled instances

- Outcome of learning process
  - transductive learning: the predictions themselves
  - inductive learning: a function that can predict the label of any unlabeled instance

- Explainable AI
  - interpretable: can be intepred
  - black-box: non-interpretable

- Learning

  - Supervised learning: from labeled
  - Unsupervised learning: from unlabeled
  - Semi-supervised learning: from a few labeled and many unlabeled
- Format of input data
  - input is often assumed to be a set of instances that are all described using the same variables (features, attributes)
  - i.i.d.: independent and identically distributed
    * tabular data (NN)
    * sequences
    * trees
    * graph
    * raw data: learning meaningful feaures from raw data
    * knowledge: inductive logic programming

1.6 Output formats, methods (predictive learning)

The **output** of a learning system is a model.

- output
  - parametrized functions
  - ocnjunctive concepts: a conjuntive concept is expressed as a set of conditions, all of which must be true
  - rule sets (if...then...else...)
  - decision trees
  - neural networks
  - probabilisitc graphical models
- search methods
  - discrete spaces - methods: hill-climbing, best-first
  - continuous spaces - methods: gradient descent
- typically
  - model structure not fixed in advanced - discrete
  - fixed model structure, tune numerical parameters - continuous
- hypothesis space
  - definition: all possible instances
  - for robot example: {B,R,M,?} x {S,T,?} x {L,W,?} x {1,2,?}

- Verson space
  - using candidate elimination
  - pros
    - can be used for discrete hypothesis spaces
    - search for all solutions, rather than just one, in an efficient manner
    - importance of generality ordering
  - cons
    - not robust to noise
    - only conjunctive concepts

2 Lecture 2: Induction of decision tree

2.1 Overview of DT

- A decision tree represents a decision procedure where
  - you start with one question
  - the answer will determine the next question
  - and repeat, untill you reach a decision
- We will usually call the questions "tests" and the decision a "prediction"
- attribute
  - input attribute X = {X_1, X_2 ..., X_n}
  - target attribute Y
  - the tree represents a function f: X -> Y
- Example: Playing Tennis Tree
  - Outlook: X_1 = {Sunny, Overcast, Rainy}
  - Humidity: X_2 = {High, Normal}
  - Wind: X_3 = {Strong, Weak}
  - Tennis: Y = {Yes, No}
  - The tree represents a function Outlook x Humidity x Wind -> Tennis
- Boolean tree
- Continuous input attributes
  - We cannot make a different child node for each possible value!
  - Solution: use comparative test -> a finite number of possible outcomes

- Type of trees
  - target attribute Y is nomial -> classification tree
  - target attribute Y is numerical -> regression tree
- **Advantages of Tree (Why tree?)**
  - Learning and using tree is **efficient**
  - Tend to have **good predictive accuracy**
  - Tree is **interpretable**

2.2 Learn trees from data

- Two tasks for DT
  - Task 1: find the smallest tree T such that $\forall$(x,f(x))$\in$D: T(x)=f(x) (meaning that only fullfill current data set)
  - Task 2: find the tree T such that for x drawn from population D, T(x) is (on average) maximally similar to f(x) (T:model tree from data set D, f(x):true function in population D)
    - loss function: l: Y_1 x Y_2 -> R (where Y_1 is predicted value, Y_2 is actual value)
    - risk R of T, the expectation of loss function, is $E_{x\sim D}[l(T(x), f(x))]$, which is needed to be minimal.
- the basic principle
  - The approach is known as "Top-down induction of decision trees (TDIDT)", or "recursive partitioning"
    - 1. start with the full data set D
    - 2. find a test such that examples in D with the same outcome for the test tend to have the same value of Y
    - 3. split D into subsets, one for each outcome of that test
    - 4. repeat this procedure on each subset that is not yet sufficiently "pure" (meaning, not all elements have the same Y)
    - 5. keep repeating until no further splits possible
- rule representation of tree
  - trees can be written as if-then-else rules
  - rules can be simplified
- Two main questions?
  - How to choose which test should be the first? (guess: attribute with minimal entropy?)
  - When to stop splitting nodes? (guess: till pure? or threshold for probability?)

2.3 Choosing the best test

- We focus on classification tree (Y is nominal)
- Information theory
  - a good test is a test that carries much information about the class
  - "entropy" or "missing information": how many bits needed, on average, to convey a piece of infomration, if an optimal encoding is used
  - **bits** <- Question!! Do not understand the bit, how is it related to entropy?
  - But whatever, the cleverest bit has been provided as below, which is called "entropy"
    - $e = -\sum_{i=1}^{k} p_i \log_2 p_i$
  - The number e reflects the minimal number of bits that you will need, on average, to encode one value. It is the inherent information content, or **entropy**.
- for classification - we use class entropy
  - The class entropy of a set S of objects($\mathbf{x}$,y), where y can be any of k classes c_i, is defined as
    - $CE(S) = -\sum_{i=1}^{k} p_i \log_2 p_i$ with $p_i = \frac{(|(x,y) \in S | y=c_i|)}{|S|}$
    - it measures how much uncertainty there is about the class of a particular instance
  - high entropy = "many possbilities, all equally likely" - not good for splitting
  - low entropy = "few possibilities, safer to conclude" - better for splitting nodes
  - entropy measures uncertainty
- information gain (IG)
  - will have the same effect of attribute entropy
  - the information gain of a question = the expected reduction of entropy by obtaining the answer to the question
  - in the case of classification trees: expected reduction of class entropy:
    - $IG(S,t) = CE(S) - \mathbb{E}(CE(S_i)) = CE(S) - \sum_{i=1}^{o} \frac{|S_i|}{|S|} CE(S_i)$
    - with t a test, o the number of possible outcomes of attribute/test t, and S_i the subset of S for which the i'th outcome was obtained.

However, if we focus on regression tree (Y is numerical), can we still use class entropy or information gain?

- Now we assume Y is numerical
- Now we use **variance reduction** instead of entropy or information gain
  - the variacne of Y in a set S of instances (x,y) is
  - $Var(S) = \frac{\sum_{(x,y)\in S}(y-\hat{y})^2}{|S|-1}$ with $\hat{y} = \frac{\sum_{(x,y)\in S} y}{|S|}$
  - and the variance reduction will be (which is similar to information gain (IG))
  - $VR(S,t) = Var(S) - \sum_{i=1}^{o} \frac{|S_i|}{|S|} Var(S_i)$

2.4 Stop splitting nodes

- In principle, keep splittign untill all instances in a subset have the same Y value
  - However, this is useful for task 1, but less useful for taks 2 (may cause overfitting problem!)
  - Please remember this is not population, this is just a sampling sample.
- overfitting
  - overfitting improves the consistency of the tree with the given data set D, but may decrease accuracy for instances outside D (whole population $\mathcal{D}$)
- How to avoid overfitting?
  - "cautious splitting": do not split a node unless you are certain that the split is meaningful
  - "post-pruning": do not bother about overfitting while splitting nodes, but once the (large) tree has been built, prune away branches that turned out not to contribute much
- "Cautious splitting"
  - How do we know when not to split a node any further?
    - a simple approach: try to guess when accuracy on unseen data is going down (**"the turning point"**)
  - But how canwe guess this turning point, if the data is unseen?
    - Solution: we can use "validation data" to evaluate
    - this "validation data" is not for growing tree, only for estimating accuracy on "unseen" data
- "Post-pruning"

  - What if the accuracy will increase again? - afraid to end the growth too early
    * Solution: we can compute the whole data set, then find its highest point
  - Princeple
    * grow the tree to its full size, then cut away branches that did not contribute to getting better predictions
    * How to decide which branch does not contribute
      · check for each node in the tree, starting at the bottom: "what would be the accuracy if I cut the tree here?" - if that accuracy is not lower, cut the tree, otherwise, do not cut.

- Pros and Cons
  - post-pruning requires more effort to build a large tree, while it gives more accurate tree
  - cations splitting is more efficient, while the accuracy is not as good as post-pruning

2.5 A generic algorithm

- TDIDT = "Top-down induction of decision trees", also referred to as "recursive partitioning"
- most decision tree learners follow the same basic approach, but differ in the details
  - we will have a look at the commonalities and differences

```
function TDIDT(E: set of examples) returns tree;
    T' = grow_tree(E);
    T = prune_tree(T');
    return T;

function grow_tree(E: set of examples) returns tree;
    T = generate_tests(E);
    t = best_test(T, E);                          (call t's outcomes v1...vk)
    P = {E1, E2, ..., Ek} with Ei={x∈E | t(x)=vi}   (P = partition induced on E by t)
    if stop_criterion(E, P)
    then return leaf(info(E))
    else
        for all Ei in P: Ti := grow_tree(Ei);
        return node(t, {(v1,T1), (v2, T2), ... (vk, Tk)});
```

Figure 1: Algorithm for "Top-down in duction of decision trees"

- The blue functions are where implementations differ

  - prune_tree: how to prune the tree afterward
  - generate_tests: which tests to consider
  - best_test: which test to select (use heuristics: entropy or variance)
  - stop_criterion: when to stop (cautious splitting or post-pruning)
  - info: what information to store in the leaf

- generate_tests
  - for numerical attributes: oblique trees can be used for determining c(threshold), while it will be more difficult even though it has higher accuracy. Thus, in practice, non-oblique trees are much more common.

- best_test
  - for classification trees: information gain (IG) = reduction to entropy $CE(S) = -\sum_{i=1}^{k} p_i \log_2 p_i$
    - in some cases: "Gini impurity" instead of entropy: $Gini(S) = 1 - \sum_{i=1}^{k} p_i^2$
  - for regression trees: reduction of variance σ, or reduction of standard deviation $\sqrt{\sigma}$
    - in some cases: normalization is impelmented by "Split information" (SI): $SI(S,t) = -\sum_{i=1}^{n} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$
    - and the Gain Ratio (GR) will be: $GR(S,t) = \frac{IG(S,t)}{SI(S,t)}$
  - for numerical inputs, how to determine c?
    - Solution: typically, all values are tried, and the c that yields the best heuristic value (IG, GR, Gini, ...) is selected (best of them)

- info
  - for classification trees: usually, the most frequent class
  - for regression trees: usually, the mean of all target values in that leaf
    - possible: median

- stop_criterion
  - 1. cautious splitting, post-pruning
  - 2. threshold
    - classification: all instances have the same class, pure
    - regression: variance
  - too few examples to continue splitting
    - only allow tests that yield subtrees with at least two examples each

- impurity measures
  - good impurity measures are strictly concave <- Question: do not understand!!

2.6 Computational complexity

- Given a data set D consisting of N instances (x,y), where x has m components (attribtues), how do N and m influence the computational effort required to learn a tree?
- "Splitting one node" - for each node, we need to "find the best test"
  - for each possible test, evaluate its quality
    - partition the dataset according to this test
    - compute quality based on this partition
  - for the test with highest quality
    - partition data according to that test
- Efficiently computing test quality for continous attribtues
  - first **sort** all tuples according to A (attribute), small to large
  - then gradually move the threshold, and simply update the tables
  - thus, testing **all thresholds** is only slightly more work than testing one!
- Computing the quality of one test
  - for nominal attributes: 1 scan, gradually building the table; compute IG once, at the end
  - for continous attributes: 1 scan, gradually updating the table; compute IG for each intermediate table
  - hence, computing the best test for a node is linear in the number of instances in that node $O(n)$, with n the numebr of instances
  - Complexity:
    - compute quality of one test is $O(n)$
    - compute quality of all tests is $O(nm)$, with m the number of attributes
- Splitting multiple nodes
  - the total at one level of the tree is always n, so the overal amount of work depends on the layer of the trees
    - if the splits are balanced, the height of tree is $O(log(N))$, and the overall compelxity of tree growth is $O(mN \log_2(N))$
    - if the splits however are unbalanced, the height of tree is $O(N)$, and the complexity is $O(mN^2)$
- For impurity measures
  - "pure" = "zero variance/entropy"

- Why decision tree for Big Data?
  - QUICK! and FAST! efficient!
  - high accuracy
  - interpretable can be also a reason

2.7 Missing values

- when computing quality of a test: just ignore instances where this value is missing
- when splitting on an attribute:
  - guess its most likely value
  - partially assign the example to multiple branch, with its probability

2.8 Model trees

Model trees are **regression tree** in which each leaf does not contain a constant, but a linear model.

- RETIS (M5)
- Mauve

2.9 Multi-target trees

- Classification and regression trees are very popular for predicting one target variable
- But the principle of variance reduction is easily generalized to predicting vectors!
- This allow us to predict multiple variables at the same time, using one tree. **(vector)**
- Multi-label classification
  - How? For example for Y = {a,b,c,d,e}
  - Option 1: "binary relevance"
    - learn (y/n) decision tree for each label
    - TreeA predicts a(y/n), TreeB predicts b(y/n)
  - Option 2: "label powersets"
    - consider each set as separate label, for 5 original labels a,b,c,d,e, we get 32 combined labels: -,a,b,c,d,e,ab,ac, ..., abc, ...
    - Learn a tree that predicts the combined label

  - Option 3: "Vector encoding"
    * encode each set as 0/1 vector, e.g. {a,b,d} = [1,1,0,1,0]
    * use a learner that can learn models that predict vectors
  - Option 3: "Vector encoding" is better! Requires almost no changes in the algorithm.
    * for example a leaf contains {[1,1,0,1,0],[1,1,0,0,0],[1,1,0,1,1]} -> [1,1,0,0.67,0.33]
    * predict all labels by threshold (e.g. 0.5): [1,1,0,0.67,0.33] -> {a,b,d}

- Hierarchical multilabel classification (HMC)
  - for protein
  - similar just add a constrant "a label can only occur in an instance's label set if all its ancestors also occur"
    * 250 functions = 250 trees is not fast and interpretable!
    * 250 functions = 1 tree is fast and interpretable!
    * learn 1 tree that predicts 250-dimensional class vector
  - Decision tree can be converted to rules set

2.10 Practical software

- Weka
- PythonL scikit-learn
- others: R, SAS, SPSS